Rust Book - 3.5: Control Flow

#rust Control Flow

Blocks of code associated with the conditions in `if` expressions are sometimes called arms, just like the arms in `match` expressions

Unlike languages such as Ruby and JavaScript, Rust will not automatically try to convert non-Boolean types to a Boolean.

Using if in a let Statement

Because `if` is an expression, we can use it on the right side of a `let` statement:

```
fn main() {
    let condition = true;
    let number = if condition {
        5
    } else {
        6
    };
}
```

The `number` variable will be bound to a value based on the outcome of the `if` expression.

```
let number = if condition {
        5
    } else {
        “six”
    };
```

variables must have a single type. Rust needs to know at compile time what type the `number` variable is, definitively, so it can verify at compile time that its type is valid everywhere we use `number`. Rust wouldn’t be able to do that if the type of `number` was only determined at runtime; the compiler would be more complex and would make fewer

guarantees about the code if it had to keep track of multiple hypothetical types for any variable.

Returning from loops

One of the uses of a `loop` is to retry an operation you know can fail, such as checking if a thread completed its job. However, you might need to pass the result of that operation to the rest of your code. If you add it to the `break` expression you use to stop the loop, it will be returned by the broken loop:

```
fn main() {
    let mut counter = 0;
    let result = loop {
        counter += 1;
        if counter == 10 {
            break counter * 2;
        }
    };
    assert_eq!(result, 20);
}
```